Протокол до Домашньої роботи №2

HAZELCAST

Андрій Полішко

Distributed map

1. Сторення та запис до розподіленої мапи

```python
import hazelcast

client = hazelcast.HazelcastClient(
cluster_name="hello-world",
)

# Create a Distributed Map in the cluster
map = client.get_map("my-distributed-map").blocking()

values = ['BMW', 'Mercedes', 'Audi', 'Toyota', 'Honda', 'Ford', 'Nissan', 'Tesla', 'Hyundai', 'Kia']

for i in range(1000):
  map.put(str(i), values[i % len(values)])
```

2. Розподіл даних у Managment Center

Member	Entries	Gets	Puts	Removals	Sets	Entry Memory	Events
127.0.0.1:5701	338	0	345	0	0	43.55 kB	0
127.0.0.1:5702	319	0	319	0	0	41.14 kB	0
127.0.0.1:5703	343	0	345	0	0	44.21 kB	0
TOTAL	1,000	0	1,009	0	0	128.90 kB	0

1 – 3 of 3 Rows 10

3. Зміни розподілу за різних обставин
   a. При відключенні однієї ноди(третьої)

Member	Entries	Gets	Puts	Removals	Sets	Entry Memory	Events
127.0.0.1:5701	514	0	345	0	0	66.21 kB	0
127.0.0.1:5702	486	0	319	0	0	62.69 kB	0
TOTAL	1,000	0	664	0	0	128.90 kB	0

1 – 2 of 2 Rows 10

   b. При відключенні двох (другої та третьої)

Member	Entries	Gets	Puts	Removals	Sets	Entry Memory	Events
127.0.0.1:5701	1,000	0	345	0	0	128.90 kB	0
TOTAL	1,000	0	345	0	0	128.90 kB	0

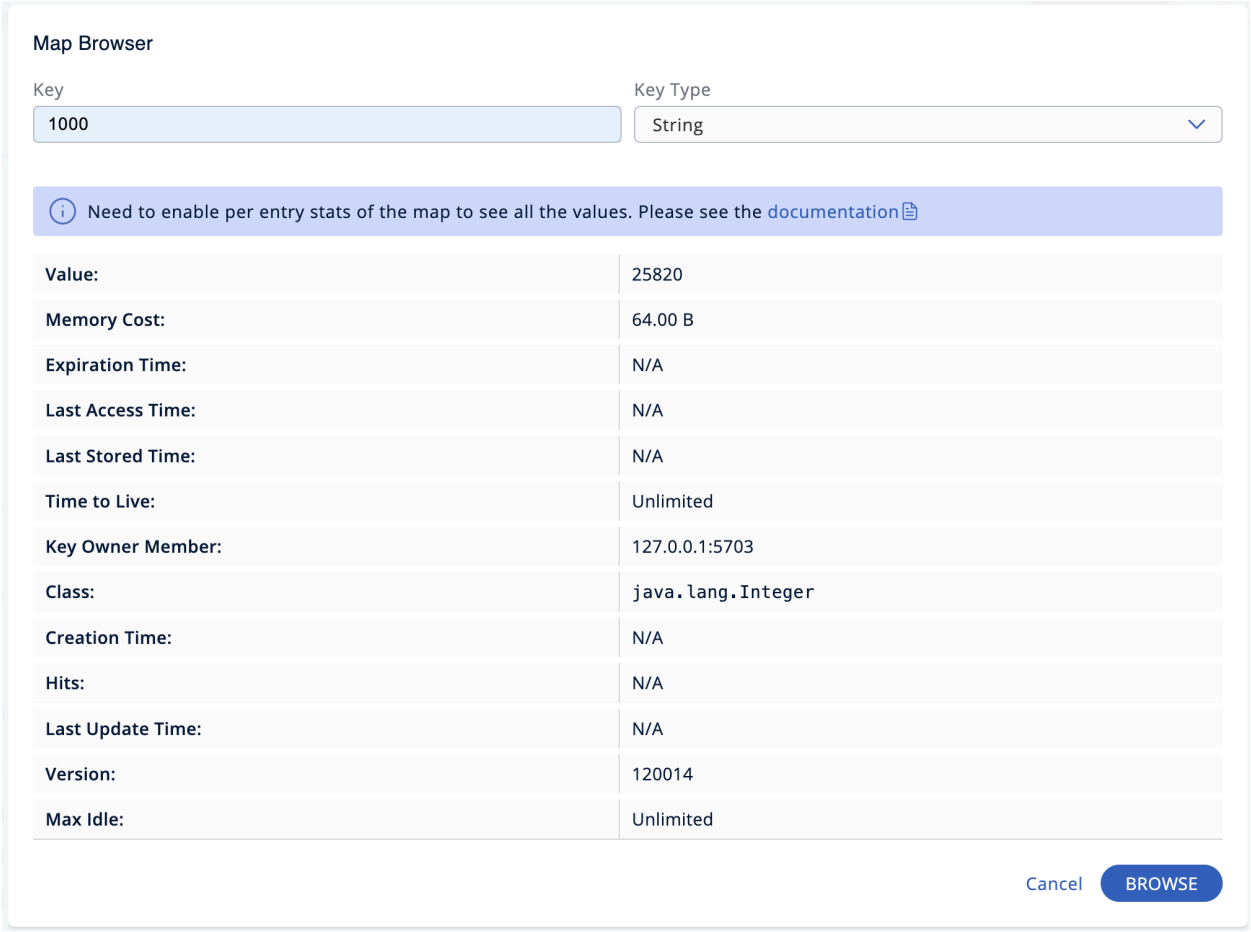
1 – 1 of 1 Rows 10

Як бачимо на останньому скріншоті, втрати даних не відбулось. Це

досягається за допомогою реплікації.

Distributed map with and without locks

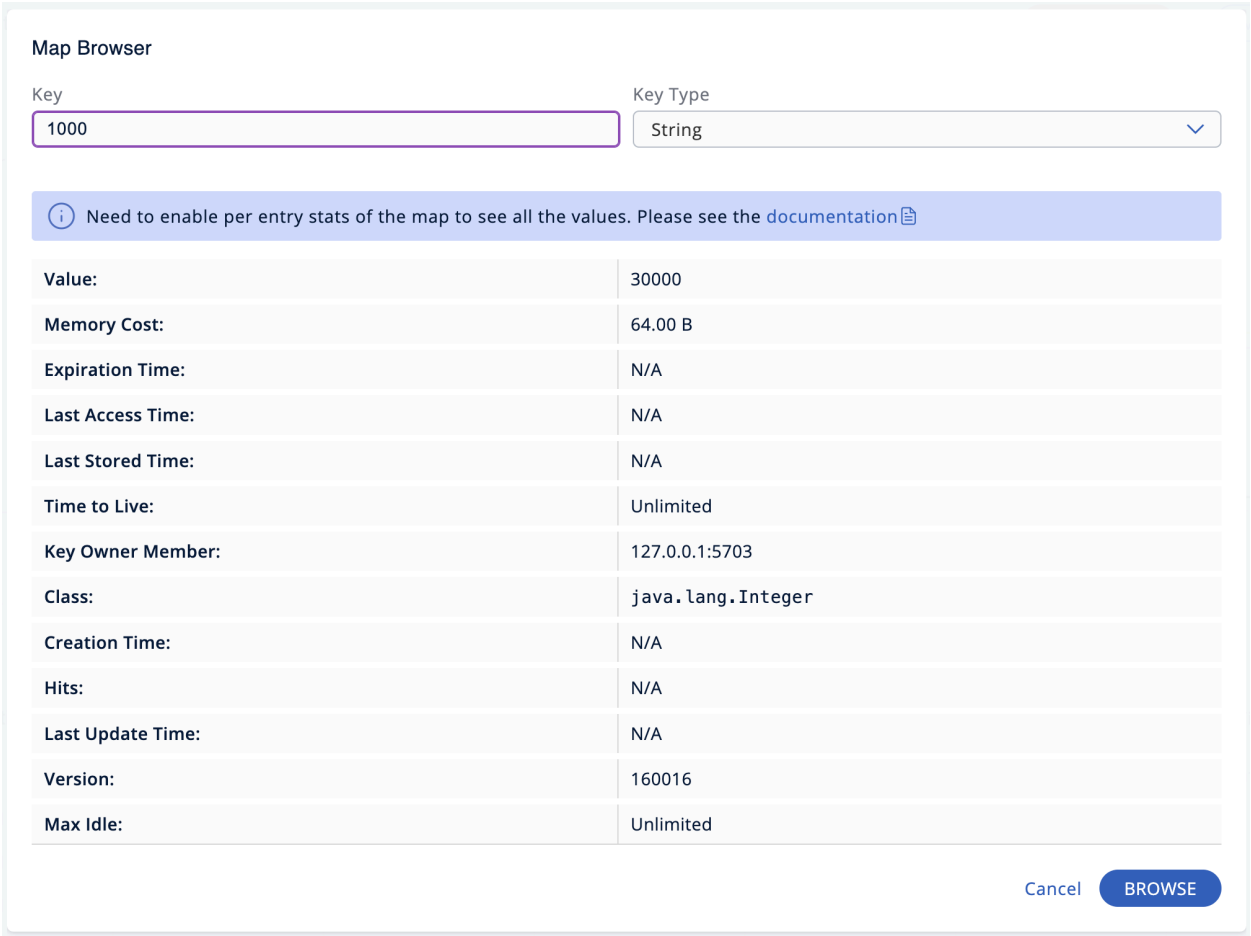
1. Значення при інкременті 10000 разів трьома клієнтами без механізмів синхронізації одночасно



Як бачимо, значення не дорівнює 30000, через race condition.

Код використаний для цього прикладу знаходиться у теці *./raceConditionTest/notLockedExamples*

2. Аналогічний тест з використанням блокування нашої змінної (Pessimistic locking)



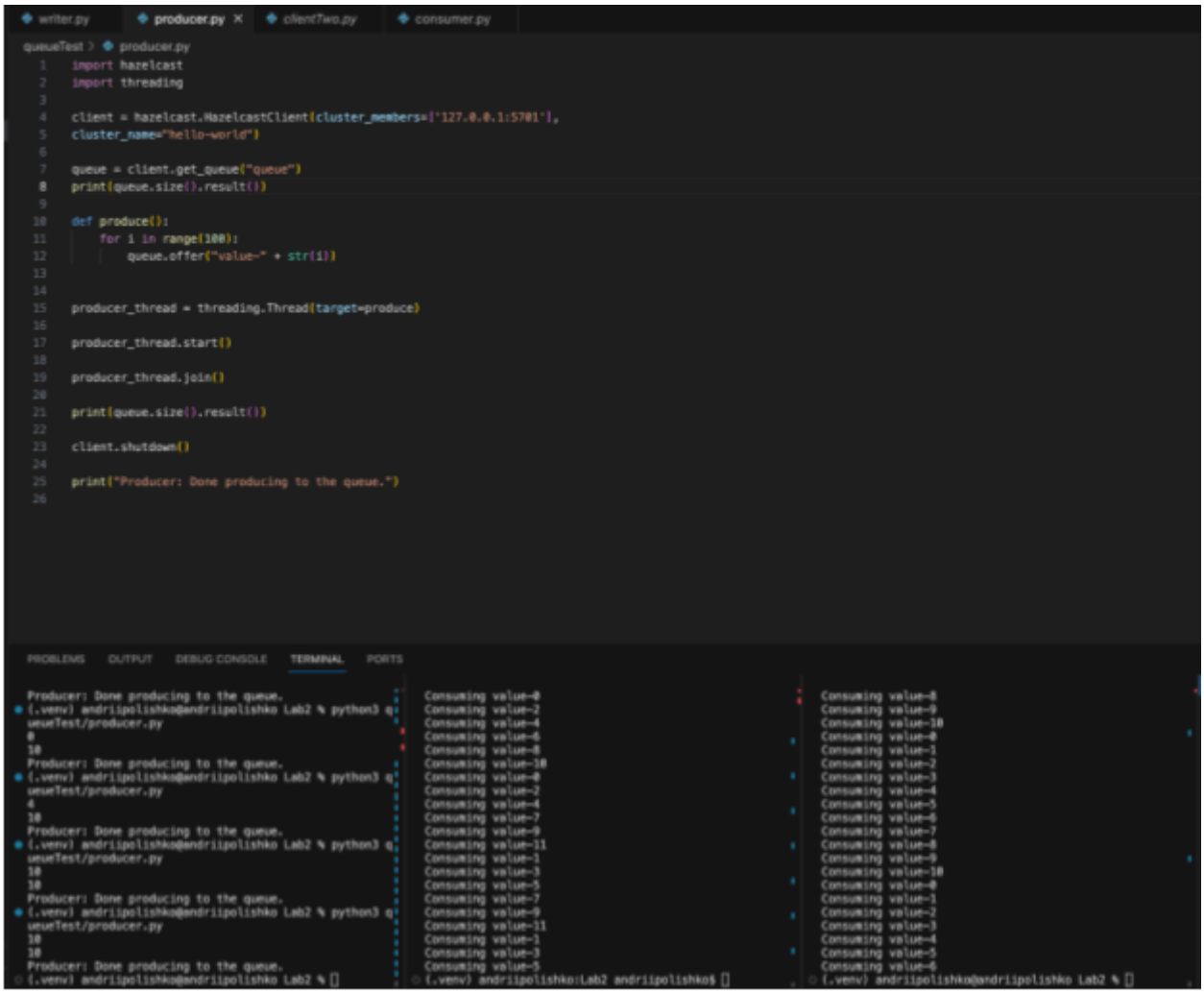
Як і очікувалося, значення дорівнює 30000, тому що при кожній зміні значення, воно блокувалось поточним клієнтом (це також відобразилось на швидкодії, такий код був відчутно повільніший)

Код використаний для цього прикладу знаходиться у теці *./raceConditionTest/pessimisticLockedExamples*

3. При оптимістичному блокуванні результат, очікувано, аналогічний

Код використаний для цього прикладу знаходиться у теці *./raceConditionTest/optimisticLockedExamples*

Bounded queue



Результати декількох запусків одного продюсера та двох конс'юмерів

Так як ми поставили побмеження на 10 елсентів до нашої черги, то відповідно запускати продюсера потрібно декілька разів, томущо кожен консюмер очікує по 100 елементів.

Конс’сюмери забирають елемети з черги по черзі, якщо запущені паралельно.

Якщо консюмери не запущені, а черга переповнилась, то просто не додаються нові елементи.

Посилання

https://github.com/AndriiPolishko/APZ_labs/tree/main/Lab2